VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Ant Colony Optimization Algorithm for Vehicle Routing Problem

Řešení problému okružních jízd s využitím optimalizace pomocí mravenčích kolonií

2017 Jan Vargovský

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Bachelor Thesis Assignment

Student: **Jan Vargovský**

Study Programme: B2647 Information and Communication Technology

Study Branch: 2612R025 Computer Science and Technology

Title: Ant Colony Optimization Algorithm for Vehicle Routing Problem
Řešení problému okružních jízd s využitím optimalizace pomocí mravenčích kolonií

The thesis language: English

Description:

The student will focus on solving Vehicle Routing Problem. The bachelor thesis will be focused on this problem by using Ant Colony Optimization (ACO). The student will implement the ACO algorithm and will study the solutions with available test benchmarks for this type of the problem. The student will also focus on visualisation for the demonstration of the ACO solution of the selected problem. It is supposed parallel implementation of the algorithm.

The detailed instructions are as follows:
1. Familiarization with the Ant Colony Optimization (ACO) algorithm.
2. ACO algorithm implementation.
3. Experimental verification of the implemented variant of ACO on a test dataset.
4. Implementation of simple visualisation of implemented variant of ACO algorithm for Vehicle Routing Problem.
5. Result assessment.

References:

[1] John E. Bella, Patrick R. McMullen: Ant colony optimization techniques for the vehicle routing problem, Elsevire, Advanced Engineering Informatics 18 (2004) 41–48.
[2] James McCaffrey: Test Run - Ant Colony Optimization, MSDN Magazine, February 2012.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Jan Martinovič, Ph.D.**

Date of issue: 01.09.2016

Date of submission: 28.04.2017

doc. Dr. Ing. Eduard Sojka
Head of Departmen

prof. RNDr. Václav Snášel, CSc.
Dean

I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, 12th April 2017 .....

I would like to thank my supervisor Jan Martinovič for the topic, advices and help throughout the thesis. I am also grateful to Ekaterina Grakova for apt remarks and invaluable feedback. Finally, I would like to thank my family and friends for all their support.

Abstrakt

Tato bakalářská práce studuje optimalizace mravenčích kolonií a jeho rozšíření. Popíšeme a naimplementujeme algoritmy lokálního vyhledávání pro problém obchodního cestujícího a problému okružních jízd s využitím optimalizace mravenčích kolonií. Výsledky algoritmu jsou poté vizualizovány pomocí vizualizačního nástroje. Algoritmus řešící problém okružních jízd s využitím optimalizace pomocí mravenčích kolonií je použit pro výpočet třech instancí. Tyto tři instance jsou také vypočítány pomocí jiných algoritmů. Výsledky jsou poté porovnány.

Klíčová slova: optimalizace pomocí mravenčích kolonií, problém okružních jízd, problém obchodního cestujícího, lokální vyhledávání, vizualizace

Abstract

This thesis studies the Ant Colony Optimization (ACO) and its extensions. We overview and implement local search algorithms for the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP) using the ACO. The results are then visualized using the visualization tool. The ACO algorithm for VRP is then used to calculate solutions for three benchmark instances. The benchmark instances are solved using other algorithms. These results are then compared.

Key Words: ant colony optimization, ACO, vehicle routing problem, VRP, travelling salesman problem, TSP, local search, visualization

Contents

List of symbols and abbreviations **8**

List of Figures **9**

List of Tables **10**

1 Introduction **12**

2 Definitions **13**
2.1 Travelling Salesman Problem 13
2.2 Vehicle Routing Problem 13

3 Ant Colony Optimization **16**
3.1 Description of how ACO works 16
3.2 ACO algorithms 19

4 ACO algorithm for VRP **22**
4.1 Candidate list 22
4.2 2-Opt 24
4.3 Path-exchange 27

5 Implementation **29**
5.1 Used technologies 29
5.2 Solution overview 29
5.3 ACO algorithm for VRP 30
5.4 Visualization 35

6 Comparison **40**

7 Conclusion **41**

Appendix **45**

A Contents of the enclosed CD **46**

B Compilation and run **47**

List of symbols and abbreviations

ACO	–	Ant Colony Optimization
ACS	–	Ant Colony System
AS	–	Ant System
CSS	–	Cascading Style Sheets
CSV	–	Comma-Separated Values
D3 or D3.js	–	Data-Driven Documents
EAS	–	Elite Ant System
GUI	–	Graphical User Interface
HTML	–	HyperText Markup Language
JSON	–	JavaScript Object Notation
LINQ	–	Language Integrated Query
MMAS	–	Max-Min Ant System
PCL	–	Portable Class Library
PLINQ	–	Parallel Language Integrated Query
SI	–	Swarm Intelligence
SVG	–	Scalable Vector Graphics
TSP	–	Traveling Salesman Problem
VRP	–	Vehicle Routing Problem

List of Figures

1 Example of solved TSP. . . . 14
2 Example of solved VRP. . . . 14
3 Input graph for ACO. . . . 17
4 ACO, pheromone initialization, step 1. . . . 17
5 ACO, first solution generation, step 2. . . . 17
6 ACO, pheromone update after first solution, step 3. . . . 18
7 ACO, second solution generation, step 4. . . . 18
8 ACO, third solution generation, step 5. . . . 18
9 ACO, pheromone overview, step 6. . . . 19
10 ACO, final graph with optimal path. . . . 19
11 Without applied candidate list. . . . 23
12 With applied candidate list with $cf = 3$. . . . 23
13 Extended candidate list, $cf = 3$. . . . 24
14 2-Opt, example 1. . . . 25
15 2-Opt, example 2. . . . 26
16 Before path-exchange. . . . 27
17 After path-exchange. . . . 28
18 Dependency graph of the solution. . . . 30
19 Program output - log. . . . 36
20 Program output - console export. . . . 37
21 Visualization - configuration input form. . . . 37
22 Visualization - arcs with pheromone. . . . 38
23 Visualization - arcs with a lot of pheromone. . . . 38
24 Visualization - toggles and legend. . . . 39
25 Visualization - everything enabled. . . . 39

List of Tables

1 Configuration parameters description. 31
2 Configuration parameters constraints. 32
3 VRP results comparison. 40

Listings

1 Configuration class 30
2 Solve method 32
3 SetInitialPheromone method 33
4 GenerateSolution method 33
5 GetNextNode method 34

1 Introduction

The TSP was first dealt with by mathematicians William Rowan Hamilton and Thomas Penyngton Kirkman in the 1800s [1]. The TSP is a special form of the Vehicle Routing Problem (VRP), that was first mentioned by Dantzig and Ramser in 1959 as the Truck Dispatching Problem [2]. The TSP and the VRP are two of the most popular combinatorial optimization problems [3, 4] which take an important part in the logistics, distribution and transportation. Although TSP has been there for almost 200 years and there are a variety of algorithms and techniques that try to solve this problem, no exact algorithm that would give us an optimal solution has been found yet. This statement also applies to VRP as well as to TSP and therefore they belong to the set of NP-hard problems [5]. The NP-hard problems are known for their computational complexity. An exact solution would run for a long time and that's why many heuristic algorithms have been proposed. The heuristic algorithms are considered as approximate and not accurate algorithms because it is not guaranteed that the solution is the best one, it might be, but it is likely to be a solution that is very close to the optimal one. One of the heuristic algorithms is the Ant Colony Optimization algorithm (ACO).

The ACO is described in several sections, from the basic idea to concrete algorithms that solve the TSP and the VRP. Section 4 overviews local search algorithms to improve existing results. Section 5 provides implementation details with a focus on the ACO algorithm for VRP. Part of the implementation section covers visualization of the solution including all the variables and states of the algorithm. The visualization is a great tool to a better understanding of how the solution was created and optionally to rerun the algorithm with adjusted parameters to get better results. Section 6 compares results of the ACO on the three VRP instances.

2 Definitions

This chapter provides definitions of used terms and techniques.

2.1 Travelling Salesman Problem

Although solving the TSP is not the main purpose of this work, we define it because TSP is basically used for each route that a vehicle has to go through and we will describe the ACO on the TSP.

In the TSP we are given a graph and we are looking for the shortest Hamiltonian cycle [6]. In other words, we are looking for a path that visits every node exactly once and the path is the minimal one.

You can see the solved TSP with 30 nodes in Figure 1.

2.2 Vehicle Routing Problem

The VRP is formally defined as a weighted oriented graph $G = (V, E)$ where the nodes are represented by $V = \{v_0, v_1, \ldots, v_n\}$, and the arcs are represented by $E = \{(v_i, v_j) : i \neq j\}$ [7]. A central depot is a special node where every vehicle starts and ends its route. The central depot is denoted by v_0 and the rest of the nodes are the customers [7]. Each customer has demand q_i, each arc has distance d_i and each vehicle has its capacity c_i. The following constraints have to be complied with:

- Each customer is visited only once by a single vehicle.
- Each vehicle must start and end its route at the depot v_0.
- Total demand of customers served by vehicle cannot exceed capacity c_i.

You can see the solved VRP with 30 nodes and 4 vehicles in Figure 2. Each route for a vehicle is represented by a different colour. If you compare Figure 1 with Figure 2, you can notice that in case of VRP we are basically solving multiple TSPs, one for each vehicle.

The VRP as was defined does not reflect all possible real-life requirements, therefore these constraints may be extended with a combination of:

- The driver of a vehicle can be restricted by a maximum time he can drive for.
- The route can have a maximum route length.
- The customer can be visited at a certain time window.
- Customer C_i has to be visited strictly before customer C_j.

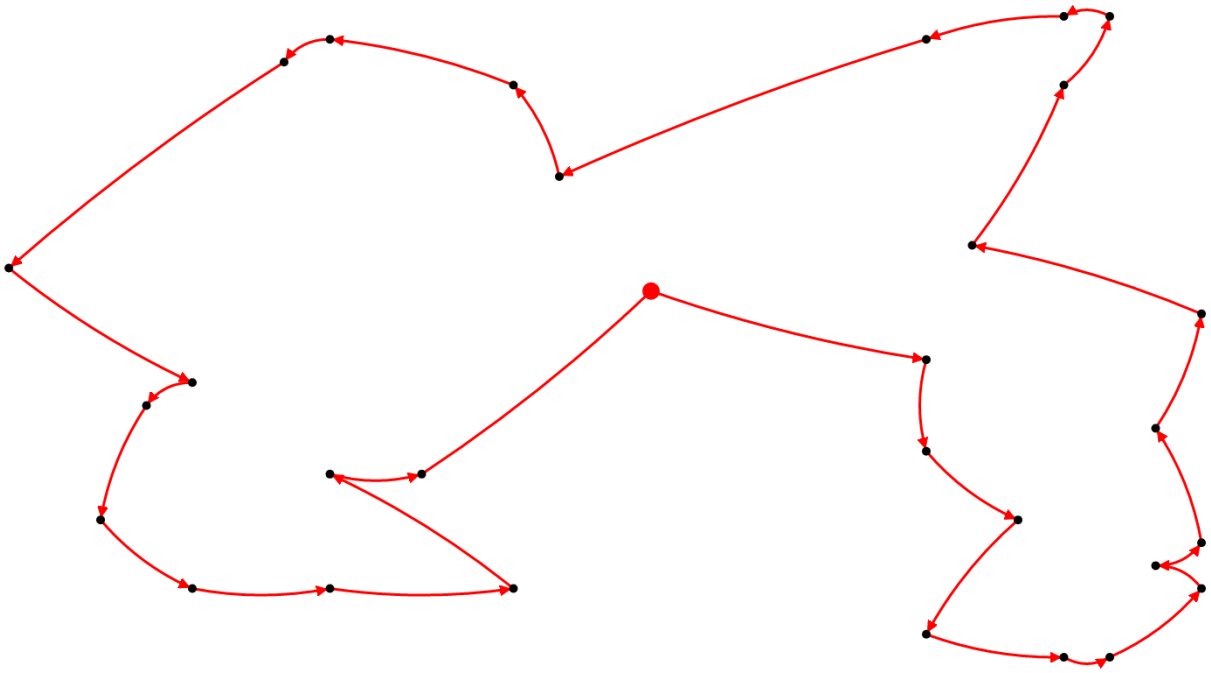


Figure 1: Example of solved TSP.

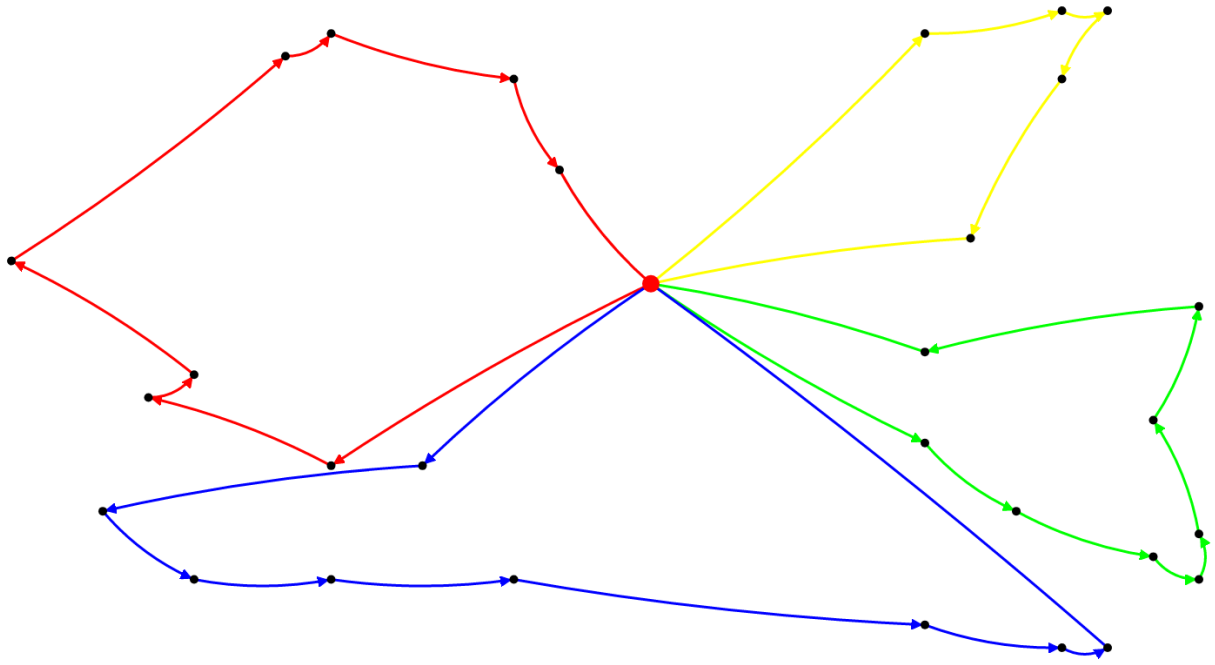


Figure 2: Example of solved VRP.

These constraints are often proposed with a proper definition and its VRP name. The most commonly known VRPs are Capacitated VRP (CVRP) [8, 9] and VRP with Time Windows (VRPTW) [9, 10]. Moreover, there are plenty of others, such as:

- Multi-Depot VRP (MDVRP) [9, 11]
- Site-Dependent VRP (SDVRP) [9, 12]
- Open VRP (OVRP) [9]
- Rich Pickup VRP with Time Windows (RPDPTW) [9]
- Periodic VRP (PVRP) [9]
- VRP with Multiple Trips (VRPMT) [13]
- VRP with Pickup and Delivery (VRPPD) [14]
- Distance-Constrained Capacitated VRP (DCVRP) [15]
- Cumulative Capacitated VRP (CCVRP) [16]

A comprehensive overview of known VRPs and its algorithms can be read in these books [17, 18].

3 Ant Colony Optimization

ACO belongs to a larger field of study called Swarm Intelligence (SI) and Bio-Inspired Computation [19, 20]. The SI algorithms are inspired by real life behaviour of multi-agent systems organised for instance by ants, bats, bees or cuckoos. These algorithms are commonly used for optimization problems, such as TSP and VRP.

The ACO is a meta heuristic technique based on the observations of the behaviour of real ant colonies looking for food. An ant's eyesight is mediocre, some of them are even completely blind, nevertheless they are still able to find food. When ants are looking for food, they just go straight forward till they find some. To find a path back, they lay down a substance called pheromone. The amount of pheromone used in a path determines how good or how bad the path is. The more amount of pheromone that is laid, the more promising the path looks. This observation was used as an inspiration by M. Dorigo, V. Maniezzo and A. Colorni in 1991 [21, 22]. What an ant does, is reducing the distance of the path from the anthill to food and vice versa. And that is exactly what we are trying to do in the optimization problems. So, they proposed a technique where ants are used to produce a solution. After the construction of a solution is done, the solution is evaluated. The pheromone is then laid down. The amount of pheromone depends on the solution's evaluation. The solution's total length is usually taken into account to evaluate the solution. ACO usually uses multiple ants and therefore there are multiple solutions constructed after each iteration. The pheromone increase is defined now, but this would lead to over-pheromoned paths after a few pheromone updates [23]. To prevent this fact, laid pheromone is evaporated with a certain percentage after the evaluation process. This pheromone evaporation gradually discards old inefficient paths. The described process is done repeatedly until the solution is suitable enough, see Pseudocode 1.

Algorithm 1 Basic ACO

```
Initialization                                   ▷ Set initial pheromone
best ← GenerateSolution
while best solution is not suitable do
    solution ← GenerateSolution
    if solution is better than best then
        best ← solution
    UpdatePheromone
return best
```

3.1 Description of how ACO works

In this section the whole process of ACO will be described. We have the following graph, see Figure 3. Arc's weight is the visual distance between the nodes. We start our procedure to calculate the shortest path between *anthill* and *food*.

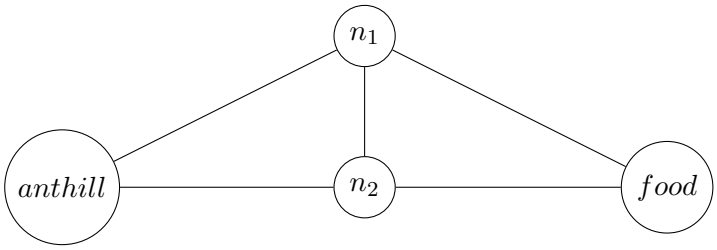


Figure 3: Input graph for ACO.

1. First we set some initial pheromone for all arcs, Figure 4.

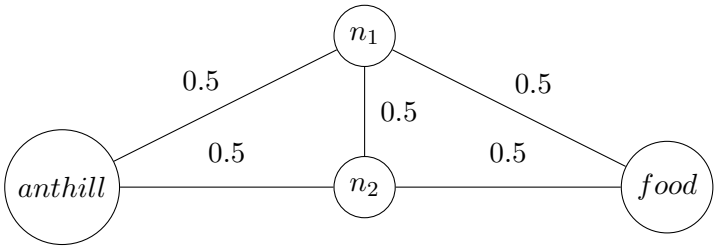


Figure 4: ACO, pheromone initialization, step 1.

2. Then we calculate a solution, Figure 5. The highlighted arcs are the solution.

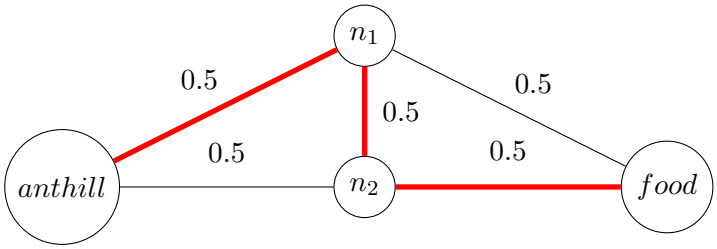


Figure 5: ACO, first solution generation, step 2.

3. We update the existing pheromone. Increase for red arcs and decrease for black ones, Figure 6.

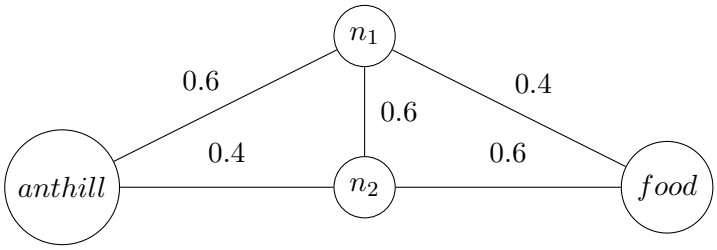


Figure 6: ACO, pheromone update after first solution, step 3.

4. We have finished our first iteration. So, let's start over and generate a new solution, Figure 7. An ant now went from *anthill* to n_1 based on the pheromone value. Then the ant decided to not follow an arc with the strongest pheromone value and try a new path instead, so he followed $food$. He found food and solution is complete.

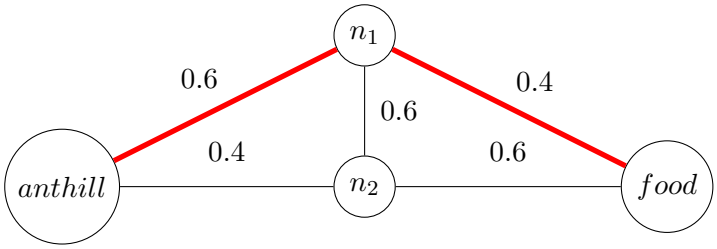


Figure 7: ACO, second solution generation, step 4.

5. A pheromone update is applied and a new solution is generated, Figure 8. We can notice that the current generated solution is far better than the previous one. It can lead to lay down more pheromone than usual, Figure 9.

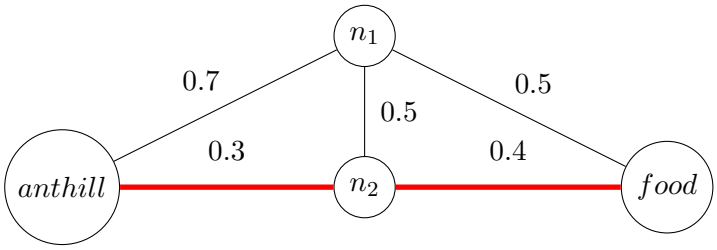


Figure 8: ACO, third solution generation, step 5.

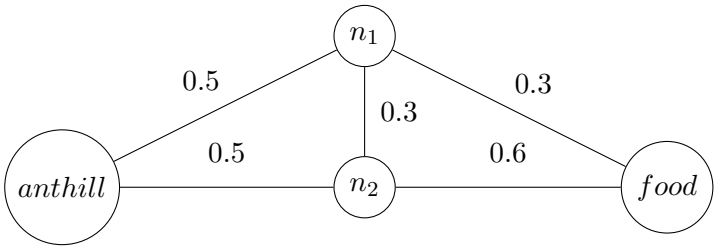


Figure 9: ACO, pheromone overview, step 6.

6. A few more iterations can be done, but no better solution would be found. The graph would converge to a final graph state. Solution's arcs are highlighted in green, Figure 10.

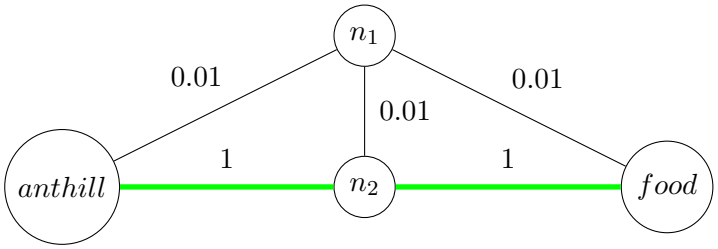


Figure 10: ACO, final graph with optimal path.

The non-deterministic decision, as we can see in Figures 7 and 8, gives ACO power to improve the existing solutions. This approach also prevents us from getting stuck because we are able to get out of the local minimum and try a new path. The existing solutions are projected into pheromone, and therefore a subset of arcs that apparently showed good solutions may be reused to create even better solutions.

3.2 ACO algorithms

The ACO is still being improved and many approaches have been proposed since then. We overview the most popular extensions here, namely the *Ant System* (AS), the *Ant Colony System* (ACS), the *Max-Min Ant System* (MMAS) and the *Elite Ant System* (EAS). The extensions are, in practise, often combined together to provide the best result.

3.2.1 Ant system

The AS was the first proposed ACO algorithm. This extension updates pheromone for all ants as follows:

$$\tau_{ij} = (1 - p) \cdot \tau_{ij} + \sum_{k=1}^{m} \triangle \tau_{ij}^{k}, \tag{1}$$

where

p is the evaporation rate, $p \in (0, 1\rangle$,

m is the number of ants,

$\triangle\tau_{ij}^k$ is the pheromone value laid on the arc A_{ij} between customers C_i and C_j by the k-th ant.

$$\triangle\tau_{ij}^k = \begin{cases} \frac{1}{L_k} & \text{if ant } k \text{ used arc } A_{ij} \text{ in its solution,} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where L_k is the solution's length by the k-th ant.

Construction of a solution for each ant is up to following equation, often called transition probability or transition rule:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in \Omega^k} \tau_{il}^\alpha \cdot \eta_{il}^\beta} & \text{if } j \in \Omega^k, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where

η_{ij} is equal to d_{ij}^{-1}, d_{ij} is the distance between customers C_i and C_j,

Ω^k is a set of nodes that are not part of the solution yet, $\Omega^k \subseteq \Omega$,

α is a user defined constant that determines importance of the pheromone,

β is a user defined constant that determines importance of the heuristic information,

p_{ij}^k is the probability to move from customer C_i to customer C_j.

Note that parameters α and β are used in contrast to provide importance of the pheromone versus heuristic information during the selection of the next node. Equation 3 calculates the probability value for each node in a graph, 0 for all nodes that we have already visited and probability p_{ij} otherwise. In practise, algorithm *Fitness proportionate selection*[1] is used to choose the next node based on the probabilities.

3.2.2 Ant colony system

The ACS improved the transition rule of the AS. Equation for the transition rule proposed in the AS is missing the ability to determine a balance between exploitation and exploration. The exploitation is a deterministic rule that simply chooses the arc with the greatest amount of the laid pheromone. The exploration is a non-deterministic (heuristic) behaviour as we have defined in Equation 3. To achieve this balance we have to introduce the following equation that

[1] Known also as *Roulette wheel selection.*

determines if either the exploitation or the exploration will be performed [24].

$$j = \begin{cases} \max_{z \in \Omega^k} \left(\tau_{iz}^{\alpha} \cdot \eta_{iz}^{\beta} \right) & \text{if } q \leq q_0 \text{ (exploitation)} \\ S & \text{otherwise (exploration)}, \end{cases} \tag{4}$$

where

q_0 is a user defined constant on the interval $\langle 0, 1 \rangle$,

q is a randomly generated number on the interval $\langle 0, 1 \rangle$,

S is the transition rule defined in Equation 3.

3.2.3 Max-Min ant system

The MMAS is another extension of the ACO. The MMAS's idea is to explicitly limit the minimum and maximum value of the laid pheromone [25]. So, after either a global or local pheromone update, pheromone value τ_{ij} is within values τ_{min} and τ_{max}. Underflow and overflow is fixed according to the following rule:

$$\tau_{ij} = \begin{cases} \tau_{min} & \text{if } \tau_{ij} < \tau_{min}, \\ \tau_{max} & \text{if } \tau_{ij} > \tau_{max}, \\ \tau_{ij} & \text{otherwise}, \end{cases} \tag{5}$$

Parameters τ_{min} and τ_{max} are user defined. Of course, condition $\tau_{min} < \tau_{max}$ must be satisfied.

3.2.4 Elite ant system

The main purpose of using elite ants is to try to search in the neighbourhood of an existing optimum. This technique differs again in the pheromone update process. The difference between an elitist ant and a classic ant is only in the amount of deposited pheromone. An elite ant will produce more pheromone than a classic ant, however according to the study [26], this technique has a bottleneck to be stuck in local optimum without a possibility to improve a global optimum.

Equation 2 can be modified as follows:

$$\triangle \tau_{ij}^{k} = \begin{cases} \frac{e}{L_k} & \text{if elite ant } k \text{ used arc } A_{ij} \text{ in its solution}, \\ \frac{1}{L_k} & \text{if classic ant } k \text{ used arc } A_{ij} \text{ in its solution}, \\ 0 & \text{otherwise}, \end{cases} \tag{6}$$

where e is user defined coefficient, $e > 1$.

4 ACO algorithm for VRP

We have talked about the VRP and variations of ACO so far. This section overviews how to solve VRP using ACO and describes algorithms and techniques that we have used in our implementation to improve our results.

The ACO algorithm for VRP does not differ from the original ACO's idea. In fact, we only have to define some rules that each ant must accomplish, in other words, when we construct a solution, we have to make sure that constraints are not violated. In our case we have the following constrains:

- Each customer is visited only once by a single vehicle.
- We have at max m vehicles, if vehicle's capacity is full, $m + 1$ vehicle may be used.
- Each vehicle has its limited capacity that cannot be exceeded.
- Each vehicle starts and ends in the depot.
- Each customer has its demand and time required to serve.
- Each vehicle cannot exceed the time limit - that is the sum of all visited customer's time and length of route converted into time based on an average velocity.

The ant starts at the depot and the route is constructed by incrementally selecting customers until one of the following conditions is evaluated as true:

1. Any constraint would be violated by adding a next customer.
2. All customers have been visited.

In case of the first condition, vehicle goes back to the depot. An ant picks the following vehicle and the construction of a new route can begin again. In case of the second condition, the solution is marked as completed and the next step of the algorithm can be proceeded.

4.1 Candidate list

A candidate list radically improves the speed of convergence to an optimum in case of big instances. The candidate list reduces potential moves of the ant (the set of nodes), where the ant can move to, based on the distance to the neighbour [27] or on a prior knowledge that might be dynamically updated during the construction [26]. The set of all available candidates is created as follows:

$$N_{cl} = N \setminus N_v \tag{7}$$

where

N is a set of all nodes,

N_v is a set of visited nodes, $N_v \subseteq N$,

N_{cl} is the final candidate list.

This gives us a set of all nodes where we can move to, and we have to reduce this set, so we define a coefficient cf (candidate fraction), $1 \leq cf \leq |N|$. We order the N_{cl} in ascending order by distance and then we take $\frac{|N|}{cf}$ nodes. For a graphical representation of the candidate list, see Figures 11 and 12.

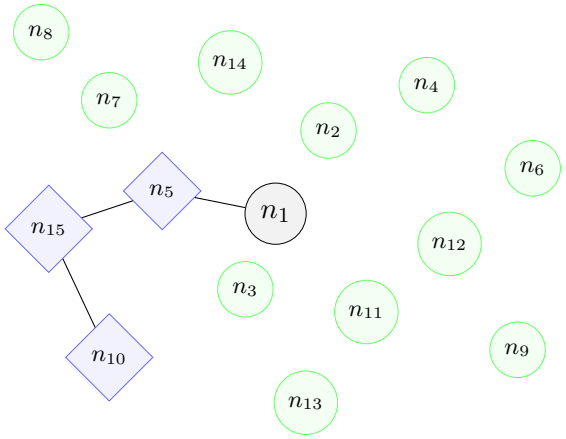


Figure 11: Without applied candidate list.

The black circle node n_1 is the current node. Blue diamond nodes are already visited nodes. The remaining, green circle nodes, are the available nodes.

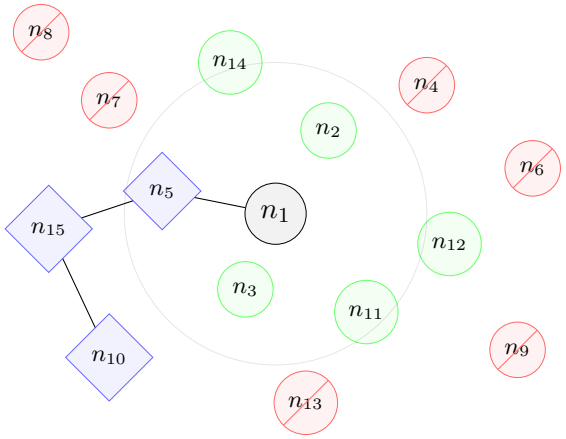


Figure 12: With applied candidate list with $cf = 3$.

Red circle nodes are nodes that are filtered out and are not considered as candidates.

We have improved the candidate list to exclude also nodes that would break any constraint. It occasionally happened that selected a node from the candidate list broke a constraint, and

therefore we had to either mark the route as invalid or try to select a new node a few times again. So, just to prevent this situation, we have extended Equation 7 as follows:

$$N_{cl} = (N \setminus N_v) \setminus N_c \tag{8}$$

where N_c is a set of not available nodes due to the violation of constraints, $N_c \subseteq N$. The result of using the extended candidate list Equation 8 instead of original one 7 can be seen in Figure 13.

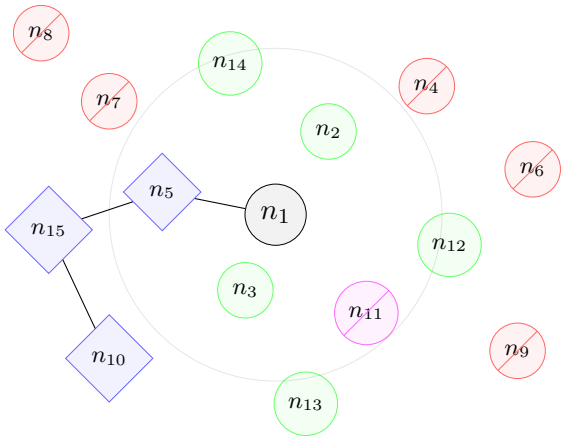


Figure 13: Extended candidate list, $cf = 3$.

The magenta circle node n_{11} is filtered out because it broke some constraint, and therefore the node n_{13} might be considered as next node now.

4.2 2-Opt

A 2-Opt algorithm is used to make a tour (TSP) 2-optimal. A tour is 2-optimal if there are no two arcs that we can remove and add two different arcs back, to reconstruct a shorter tour[2]. The 2-Opt algorithm is a heuristic algorithm to improve the route of each vehicle - the route of a vehicle represents TSP. We use this algorithm as a local search [28], in general it is considered as post-optimization process.

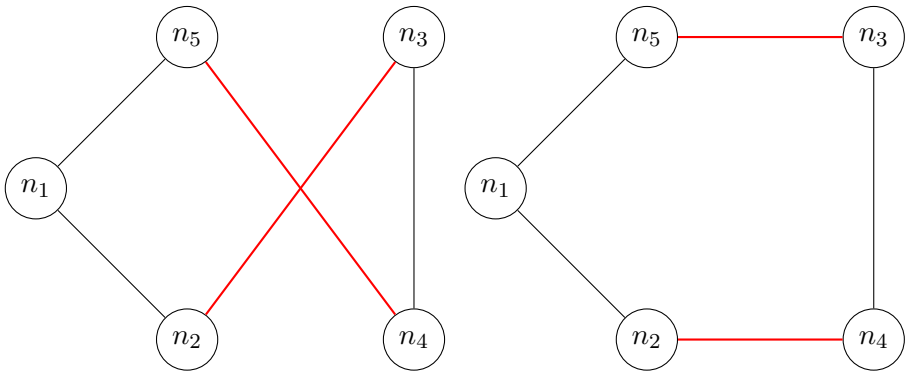
The 2-Opt algorithm just divides the input array into three groups and reverses the middle one, then it compares if the result improved the previous result, and if so, then it starts over, see algorithm 2. Two graphic examples can be seen in Figures 14 and 15.

A 3-Opt, as well as a 4-Opt versions exist [29]. The 3-Opt removes 3 arcs and adds 3 arcs back, the 4-Opt removes 4 arcs and adds 4 arcs back. This increment continues, and in general, the algorithm is called K-Opt [29]. The K-Opt removes k arcs and adds different k arcs back. However the difference of the results between the 2-Opt and the 3-Opt is negligible compared to consumed the compute time, and therefore we stick to the 2-Opt in our implementation only.

[2] `https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/lecture-notes/MIT15_053S13_lec17.pdf`

Algorithm 2 2-Opt

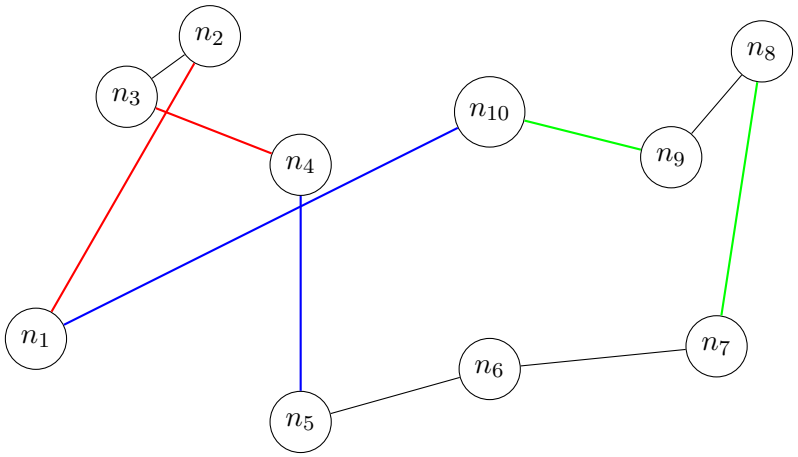
```
1:  function TwoOpt(Node[] route)
2:      best ← route
3:      n ← count(route)
4:  start:
5:      i ← 0
6:      for i < n − 1 do
7:          j ← i + 1
8:          for j < n do
9:              current ← TwoOptSwap(best, i, j)
10:             if routeLength(current) < routeLength(best) then
11:                 best ← current
12:                 goto start
13:             j ← j + 1
14:         i ← i + 1
        return best
15: function TwoOptSwap(Node[] route, int i, int j)
16:     result ← copy(route)
17:     while a < b do
18:         tmp ← route[a]
19:         route[a] ← route[b]
20:         route[b] ← tmp
21:         a ← a + 1
22:         b ← b − 1
        return result
```



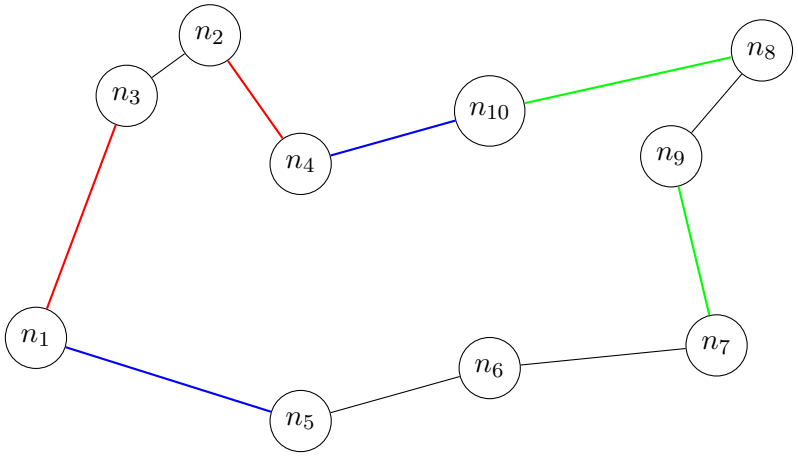
(a) Before applied 2-Opt (b) After applied 2-Opt

Figure 14: 2-Opt, example 1.

We can imagine a graph represented as an array $[n_1, n_2, n_3, n_4, n_5]$, we divide it to three sub-arrays $[n_1, n_2]$, $[n_3, n_4]$ and $[n_5]$, we reverse the middle sub-array $[n_3, n_4] \rightarrow [n_4, n_3]$ and finally we join all sub-arrays back into one. $[n_1, n_2]$, $[n_4, n_3]$, $[n_5] \rightarrow [n_1, n_2, n_4, n_3, n_5]$.



(a) Before applied 2-Opt



(b) After applied 2-Opt

Figure 15: 2-Opt, example 2.

Improvements are done in the following order:

$$[n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}] \rightarrow [n_1, n_3, n_2, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}]$$
$$[n_1, n_3, n_2, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}] \rightarrow [n_1, n_3, n_2, n_4, n_{10}, n_9, n_8, n_7, n_6, n_5]$$
$$[n_1, n_3, n_2, n_4, n_{10}, n_9, n_8, n_7, n_6, n_5] \rightarrow [n_1, n_3, n_2, n_4, n_{10}, n_8, n_9, n_7, n_6, n_5]$$

4.3 Path-exchange

A path-exchange is similar to the 2-opt. The difference is that the path-exchange is done on two arbitrary routes unlike the 2-opt, which is done on one route only. We are given two different routes N_p and N_q. Then two sets of nodes p and q are selected, $p \subseteq N_p$ and $q \subseteq N_q$, note that either p or q may be empty. The selected nodes p and q are exchanged, see Equation 9. An example of the path-exchange can be seen in Figures 16 and 17. Algorithms for the path-exchange can be read at [30].

$$\begin{aligned} N'_p &= (N_p \setminus p) \cup q \\ N'_q &= (N_q \setminus q) \cup p \end{aligned} \tag{9}$$

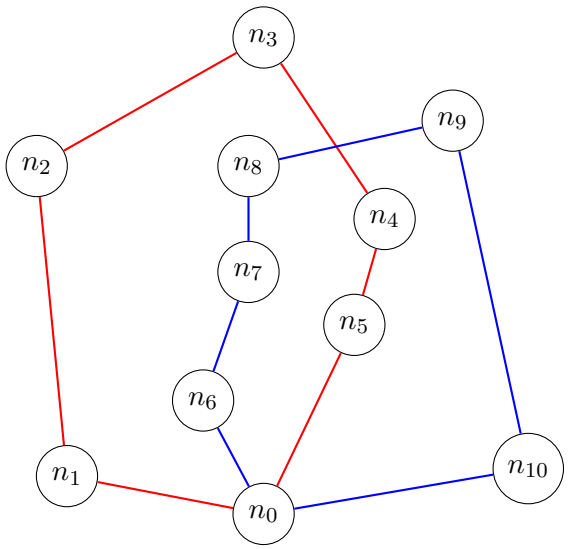
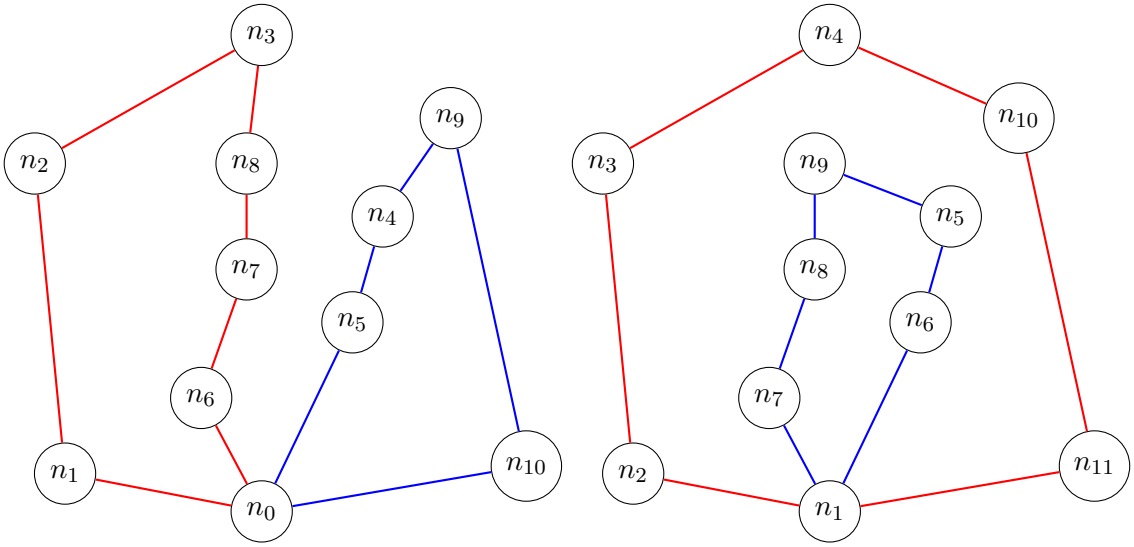


Figure 16: Before path-exchange.



(a) After path-exchange, option 1

(b) After path-exchange, option 2

Figure 17: After path-exchange.

5 Implementation

This section covers technical details of the implementation. We overview programming languages, technologies and frameworks that were used in the thesis. Then we describe our implementation of the ACO algorithm for VRP and finally, we overview our implementation for visualization, that is special for ACO and helps us to better understand how the results were created.

5.1 Used technologies

The major technology that was used is .NET Core[3]. It is a free, open-source, modular and cross-platform framework. We used it together with a Visual Studio[4] in versions 2015 and 2017.

The algorithm itself is programmed using C# 7. Server side of the visualization runs on ASP.NET Core MVC[5], which is a web framework on the top of .NET Core and ASP.NET Core[6]. Significant technology on client side is a Data-Driven Documents[7] (D3 or D3.js) library that is used to visualize the output from the ACO algorithm using HTML, CSS and SVG, which you could already see in Figures 1 and 2.

5.2 Solution overview

The solution consist of the following projects:

- **Common** - Portable Class Library (PCL) project that contains shared classes (such as Comma-Separated Values (CSV) reader and generic graph representation as adjacency matrix) that are used in other projects.
- **TSP** - This project was used for learning how ACO works on TSP. There are three implementations and four demo programs. Implementations always inherit from the previous versions and they are as follows:
  1. ACO algorithm for TSP by James McCaffrey published here [31].
  2. Complete rewrite with inspiration from original code - extract to classes, non-generic version.
  3. Second version - completely in generic with a parallel approach.

  The four demo programs run the previous three versions. The last one is run twice, once with data loaded from CSV files and once with dummy (pseudo random) data.

[3] `https://github.com/dotnet/core`
[4] `https://www.visualstudio.com`
[5] `https://github.com/aspnet/Mvc`
[6] `https://github.com/aspnet`
[7] `https://d3js.org`

- **VRP** - It is an implementation of ACO algorithm for VRP, that is described in the following section 5.3.
- **VRP.Benchmark** - This is a test project to produce statistics. It is used to generate multiple configurations that are passed in the VRP algorithm and then to run it multiple times.
- **WebHost** - It is a visualization project, that is described in the following section 5.4.

Dependency graphs of the projects can be seen in Figure 18.

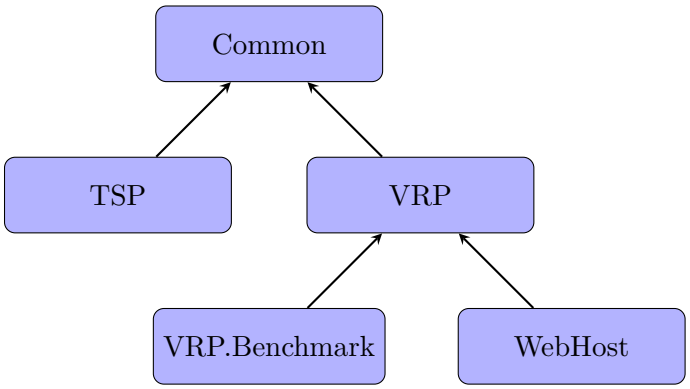


Figure 18: Dependency graph of the solution.

5.3 ACO algorithm for VRP

The implementation of the ACO algorithm for VRP is using all the theory what we have talked about in the past sections. This section provides mainly a description of the VRP project and the whole process of how our program and algorithm work.

5.3.1 Parameters

Our implementation has many parameters that can be modified. They are encapsulated in the Configuration class, see Listing 1.

```
public sealed class Configuration
{
   public int AntCount { get; }
   public int Iterations { get; }
   public double Alpha { get; }
   public double Beta { get; }
   public double P { get; }
   public double Q { get; }
   public double Min { get; }
```

```
    public double Max { get; }
    public double InitialPheromone { get; }
    public int CandidateFraction { get; }
    public double Q0 { get; }
}
```

Listing 1: Configuration class

We have not talked about all the parameters yet. There are two tables that briefly describe parameters and their constraints. Table 1 describes what every parameter in code means and its equivalent name in theory, default value and short description. Table 2 describes the constraints.

Code name	Theory name	Brief description
AntCount	m	Number of ants.
Iterations	it	Number of iterations.
Alpha	α	Importance of pheromone.
Beta	β	Importance of heuristic information.
P	p	Evaporation rate.
Q	Q	Constant to adjust laid pheromone.
Min	τ_{min}	Minimum pheromone.
Max	τ_{max}	Maximum pheromone.
InitialPheromone	τ_0	Initial pheromone value.
CandidateFraction	cf	Candidate fraction.
Q0	q_0	Exploitation versus exploration.

Table 1: Configuration parameters description.

5.3.2 Input data

The input data are stored in files in the CSV format. We need to load three types of files:

1. Nodes (Id, Latitude, Longitude, Demand, Time).
2. Arcs (From, To, Distance).
3. Vehicles (Id, Capacity).

Nodes and Arcs are combined together and are encapsulated in the adjacency matrix. Vehicles are stored in a separate collection.

Code name	Theory name	Default value	Constraints
AntCount	m	8	$m \in \mathbb{Z}^+$
Iterations	it	500	$it \in \mathbb{Z}^+$
Alpha	α	1	$\alpha \in \mathbb{R}^+$
Beta	β	2	$\beta \in \mathbb{R}^+$
P	p	0.001	$p \in (0, 1\rangle$
Q	Q	5000	$Q \in \mathbb{R}^+$
Min	τ_{min}	0.0001	$\tau_{min} < \tau_{max} \in \mathbb{R}^+$
Max	τ_{max}	1	$\tau_{min} < \tau_{max} \in \mathbb{R}^+$
InitialPheromone	τ_0	0.001	$\tau_{min} \leq \tau_0 \leq \tau_{max}$
CandidateFraction	cf	3	$cf \in \mathbb{Z}^+$
Q0	q_0	0.2	$q_0 \in \langle 0, 1\rangle$

Table 2: Configuration parameters constraints.

5.3.3 Algorithm

When the input data are loaded, they are passed into the Solve method located in the major class called VRPSolver. Important lines from the Solve method can be seen in Listing 2, note that the method is similar to Algorithm 1. The method returns a collection of solutions. The first solution is the initial one and then at the end of every iteration there is a comparison against the best solution and iteration best solution. If a better solution was found in the iteration, it is stored and yielded[8].

```
public IEnumerable<Ant> Solve(Graph graph, Configuration configuration, Vehicle[]
    vehicles)
  SetInitialPheromone(graph, configuration);
  var ants = ConstructSolutions(graph, configuration, vehicles);
  LocalSearch(ants);
  var bestAntOverall = GetBestAntByTotalDistance(ants);
  yield return bestAntOverall;

  for (int i = 0; i <= configuration.Iterations; i++)
  {
    ants = ConstructSolutions(graph, configuration, vehicles);
    LocalSearch(ants);
    UpdatePheromones(ants, graph, configuration);
```

[8] https://msdn.microsoft.com/en-us/library/9k7k7cf0.aspx

```
      var bestAntThisIteration = GetBestAntByTotalDistance(ants);
      if (bestAntOverall > bestAntThisIteration)
      {
         bestAntOverall = bestAntThisIteration;
         yield return bestAntOverall;
      }
   }
```

Listing 2: Solve method

The SetInitialPheromone method just iterates through all the arcs and sets the pheromone value to the initial pheromone (τ_0) from the configuration instance, see Listing 3.

```
private void SetInitialPheromone(Graph graph, Configuration configuration) =>
   graph.Arcs.ForEach(t => t.Data.Pheromone = configuration.InitialPheromone);
```

Listing 3: SetInitialPheromone method

Then, we call the ConstructSolutions method. This method is responsible for constructing solutions. The method creates m ants and each ant is passed to its thread[9]. Each thread executes the GenerateSolution method, see Listing 4. This approach is called the Parallel Ants, there are at least four more approaches of parallelism described here [32]. The method simulates one complete VRP solution that is created by one ant. The ant always chooses one vehicle and constructs its route, when there are no more nodes that it can visit or any constraints would be violated, it returns back, the route is stored and the ant chooses the following vehicle.

```
private Ant GenerateSolution(Graph graph, Configuration configuration, Vehicle[]
    vehicles)
{
   var routes = new List<Route>();
   HashSet<Node> visited = new HashSet<Node>
   {
      graph[0] // Depot
   };
   int vehicleIndex = 0;
   while (!AreAllVisited(visited, graph) && vehicleIndex < vehicles.Length)
      routes.Add(GenerateRoute(visited, graph, configuration,
          vehicles[vehicleIndex++]));
   return new Ant(routes);
}
```

Listing 4: GenerateSolution method

The GenerateRoute method returns a route for one concrete vehicle. The method internally iterates through the loop until a route is constructed via the GetNextNode method. When a

[9]Using the Parallel LINQ (PLINQ) library.

node is returned, it adds it to the set of visited nodes and also to the route. The GetNextNode method can be seen in Listing 5.

```
private Node GetNextNode(Node from, HashSet<Node> tabu, Graph graph, Configuration
    configuration, Route route)
{
    var candidates = from.ArcsOut
      .Where(t => !tabu.Contains(t.To))
      .Where(t => route.CanAdd(t.To))
      .OrderBy(t => t.Data.Distance)
      .Take(graph.Size / configuration.CandidateFraction);

   if (candidates.Length == 0)
      return null;

   double q = random.NextDouble();
   // Formula 1
   if (q <= configuration.Q0)
   {
      var exploitationNodes = candidates
         .Select(arc => new
         {
            Node = arc.To,
            Value = CalculateExploitationValue(arc, configuration),
         });

      // return value based on pheromone and distance
      return exploitationNodes.MaxBy(t => t.Value).Node;
   }
   // Formula 2
   else
   {
      var explorationNodes = candidates
         .Select(arc => new
         {
            Node = arc.To,
            Value = CalculateExplorationValue(arc, configuration),
            Arc = arc,
         });

      // randomly chosen (Roulette wheel selection / Fitness proportionate selection)
      var sum = explorationNodes.Sum(t => t.Value);
      double wheelPosition = 0d;
      foreach (var item in explorationNodes)
      {
```

```
            wheelPosition += (item.Value / sum);
            if (wheelPosition >= q)
                return item.Node;
        }
        return null;
    }
```

Listing 5: GetNextNode method

The candidates variable is a collection that represents the candidate list described here 4.1. It is created as follows: we have all the candidates, we remove the already visited nodes and nodes that would break any constraints rule. This gives us all the possible candidates, although take we only a fraction of them. The transition rule is then performed according to Equation 4. The methods CalculateExploitationValue and CalculateExplorationValue are calculating values according to Equations 3 and 4.

We have constructed the solutions and the LocalSearch method is executed. We pass all solutions in and the 2-Opt algorithm is performed. After that the UpdatePheromones method is executed. The implementation of the UpdatePheromones method is based on Equation 1. There is one change that we have done. When we increase a laid pheromone, we are not using Equation 2. We have updated $\frac{1}{L_k}$ to $\frac{Q}{L_k}$. Q is a configuration coefficient to adjust a laid pheromone. It helped us distinguish good and bad arcs in the visualization, that will be described in the following section. This process is repeated *it* times.

5.3.4 Output

The program output has multiple different formats. The first one is an informational log directly to standard output. The log is done before/after each important action, such as new (better) solution has been found or iteration of ACO algorithm ended, see Figure 19. The second output prints detailed report of the best solution. The report prints details of all vehicles, it means: id, list of customers and their demand, used and total capacity, total distance of the route, see Figure 20. The third and last output format is a JavaScript Object Notation (JSON) file that is used as an input to the visualization tool - it consist of nodes, arcs, vehicles, configuration and total distance of the best solution.

5.4 Visualization

The visualization is a web application. It is a Graphical User Interface (GUI) extension to the VRP project to provide better experience with configuring right parameters to the algorithm, see Figure 21.

It is sometimes hard to understand how the algorithm behaved, and therefore there is a possibility to render all the arcs in the graph. Each arc has its width and opacity scaled with an amount of pheromone value. So, an arc with a low amount of pheromone will be almost

```
AntCount: 8
Iterations: 500
Alpha: 1
Beta: 2
P: 0,001
Q: 5000
Min: 0,0001
Max: 1
InitialPheromone: 0,001
CandidateFraction: 9
Q0: 0,2
NodeRepeatCount: 1

Initial best distance is 212299,6
Nothing was found at 0/500
New best distance is 208841,4 found at 1/500
New best distance is 197238,3 found at 2/500
Nothing was found at 7/500
New best distance is 187399,5 found at 8/500
Nothing was found at 12/500
New best distance is 182391,9 found at 13/500
Nothing was found at 131/500
New best distance is 177621,3 found at 132/500
Nothing was found at 215/500
New best distance is 171632,0 found at 216/500
Nothing was found at 500/500

Best solution is 171632,0
```

Figure 19: Program output - log.

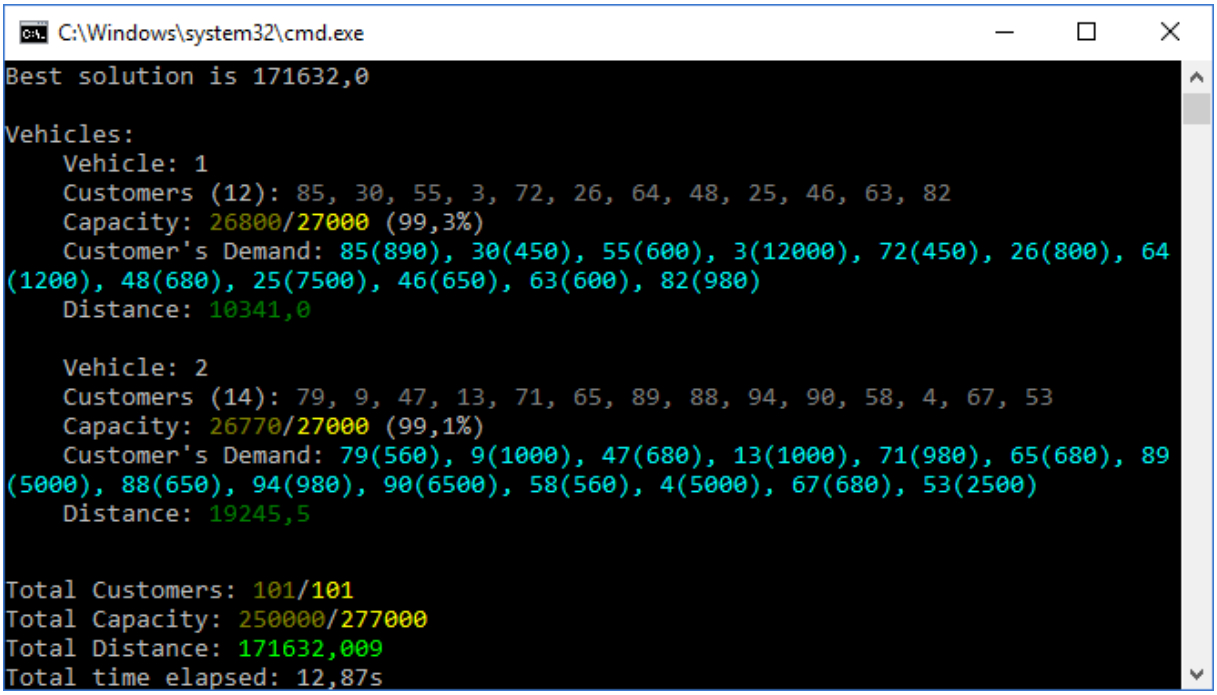


Figure 20: Program output - console export.

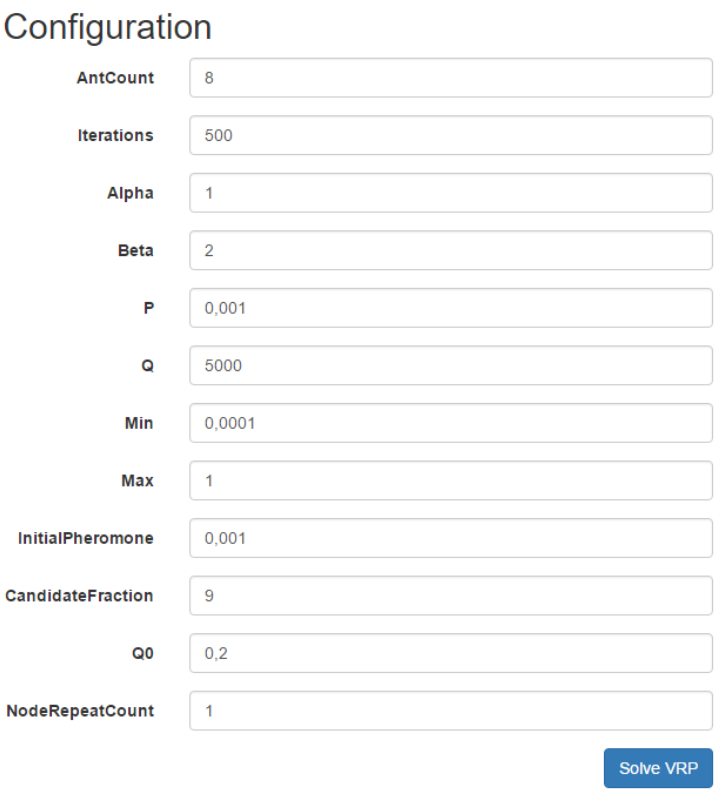


Figure 21: Visualization - configuration input form.

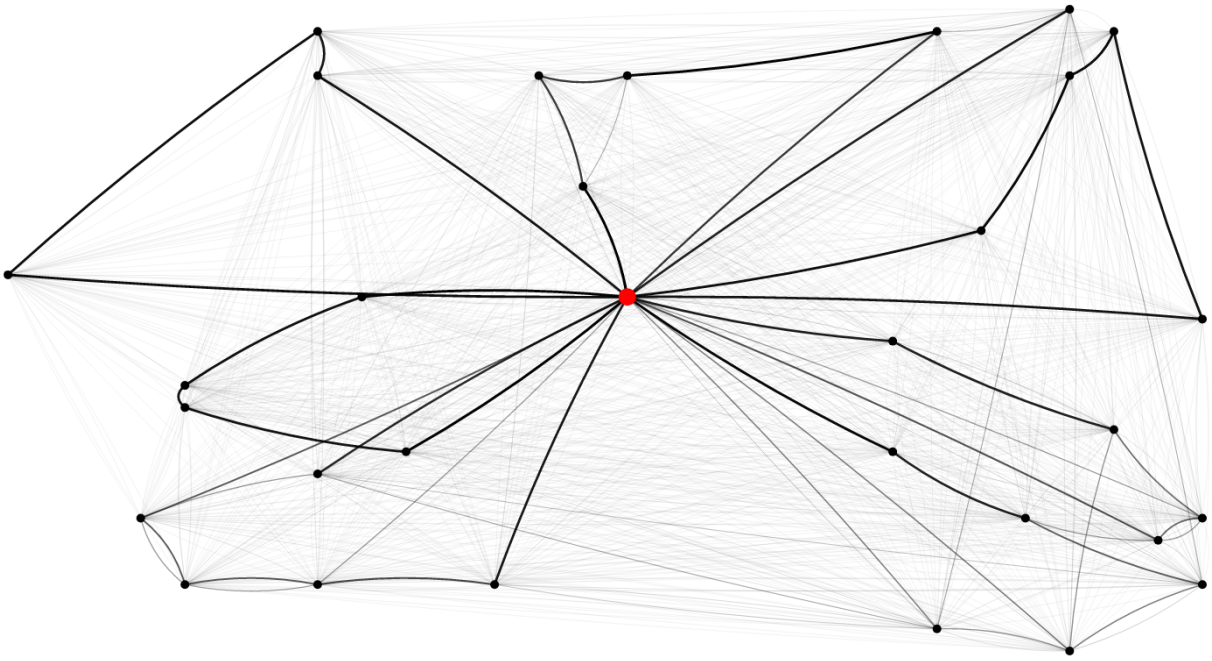
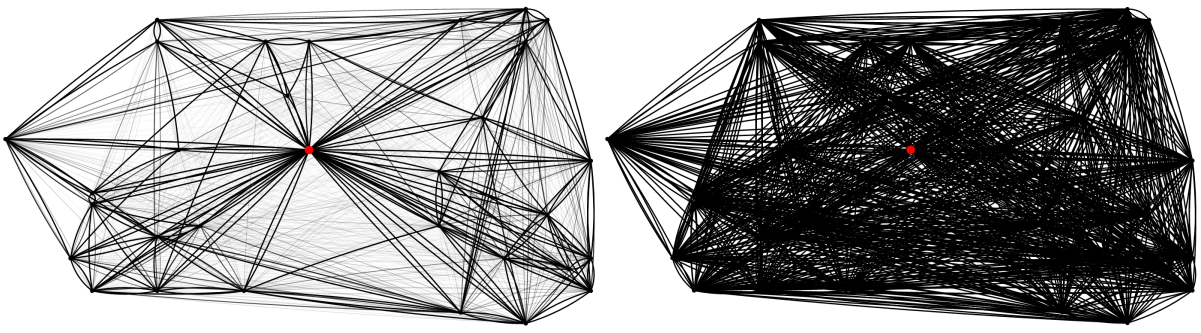


Figure 22: Visualization - arcs with pheromone.

transparent and really thin - almost invisible, on the other hand, an arc with a high amount of pheromone will be thick and clear, see Figure 22.

Two really clear illustrations of over-pheromoned arcs can be seen in Figure 23. An invalid configuration of the algorithm does not necessarily mean that it will give you a bad result. These two examples were not far from the optimum, it has to be remarked that these examples had $\tilde{3}0$ nodes only, so even if ants are going totally random, they sometimes find quite good solutions. The difference in configurations is noticeable on larger instances, let's say with instances that contains more than 100 nodes.

So, you can show all arcs and the arcs that are part of the best solution. You can also



(a) A little over-pheromoned arcs

(b) A lot over-pheromoned arcs - disaster

Figure 23: Visualization - arcs with a lot of pheromone.

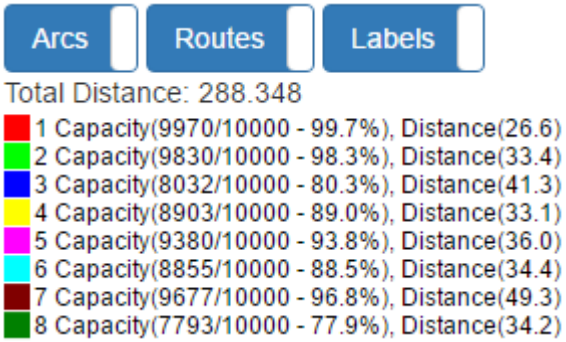


Figure 24: Visualization - toggles and legend.

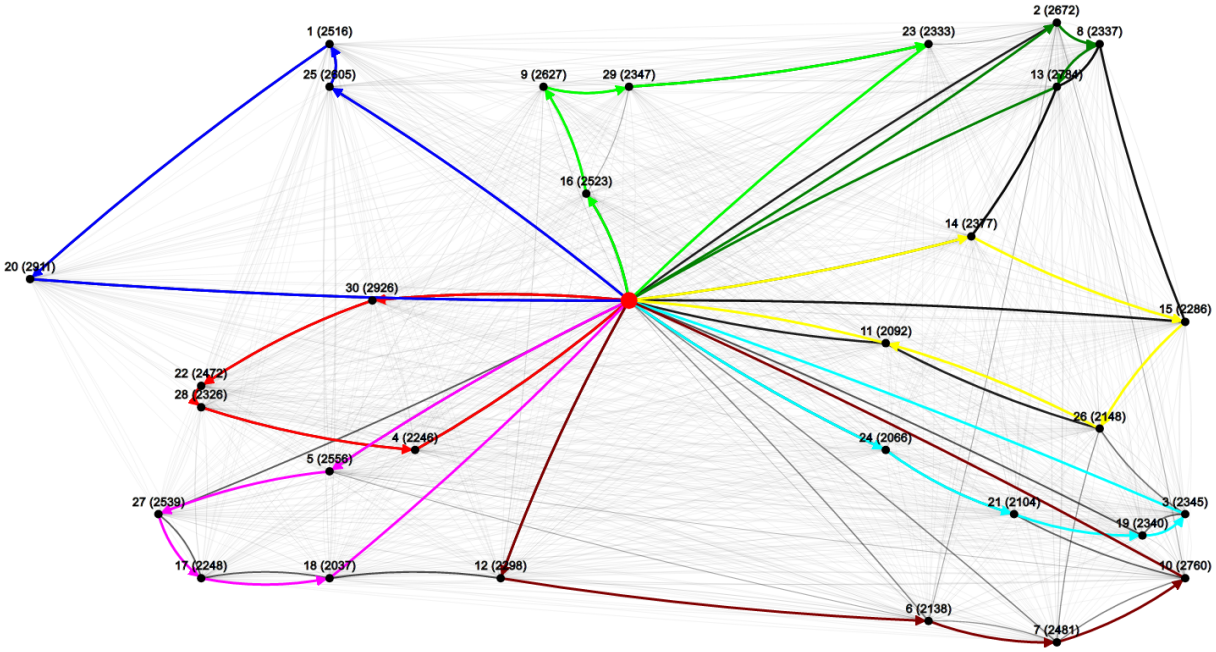


Figure 25: Visualization - everything enabled.

show a label above each node in format "Id (Demand)". There are three toggles in the top left corner, that change the visibility of each part. Under the toggles, there is a legend that contains information of each vehicle, it consists of similar information as the console export[10]. Figure 24 shows the legend and Figure 25 shows all visible information at once. If any vehicle is under the hover, it will outshine all other routes, so the route of the hovered vehicle is easy to find. It is handy in case of big problems that have either many nodes or many vehicles.

The visualization shows the last calculated VRP as default - last modified file, to be precise. There is a drag and drop feature that allows you to visualize any VRP solution, just drag a file from file system and drop it directly to the page, it will re-render the visualization.

[10]Almost all information are stored as attribute on each element, including amount of pheromone value, node id, latitude and longitude and others.

6 Comparison

We compare the ACO algorithm for VRP that was described in Section 5.3 against the algorithm that is using Adaptive Large Neighborhood Search (ALNS) [33, 34, 35] and an exact algorithm.

For the benchmark we have one benchmark instance with 101 customers and 10 vehicles (101/10). This instance is used to form two additional smaller instances. The first instance has 50 customers and 6 vehicles (50/6). The second instance has 25 customers and 4 vehicles (25/4). Note that the instances do not count depot as a customer.

The results are presented in Table 3.

Instance*	ACO[11]	ALNS	Exact*****
25/4	**63162.88****	64365**	66349,51
50/6	91191,02***	**90501*****	114169,52
101/10	**163317.98******	171891******	219654,79

* - Customers/Vehicles, ** - Runtime 5s, *** - Runtime 10s, **** - Runtime 15s, ***** - Runtime 1h.

Table 3: VRP results comparison.

We can see that the ACO provided similar results together with the ALNS in the 25/4 and 50/6 benchmark instances with difference up to 2%. The 101/10 benchmark instance shows a significant difference (5%) on behalf of the ACO. The ACO tends to provide good results in a few seconds, however the required time grows exponentially. On the other hand, the ACO still requires a lot less computation time to get good results compared to the exact algorithm that did not even provide a good result after one hour of computation. The only relevant comparison is against the results provided by ALNS algorithm and the ACO is slightly better. Unfortunately, we do not know the optimum results, so we can only assume that the ACO results are good.

Although the ACO runs relatively for a short time, we had to run it several times with different configurations. Some configurations are just totally wrong and you do not even have a clue about it, until you check the visualized result and the laid pheromone on the arcs.

There is a VRP community that gathers input data for the most known benchmark instances[12]. Each benchmark instance has its best known result and information if it is an optimum or not. The benchmark instances are up to 1200 customers. The VRP community also organises a CVRP challenge for exact methods with a prize from 300 to 500 US dollars.

[11] Tests ran on Intel Core i5-3570k@3,8GHz with 8GB RAM and Windows 10 Pro 64-bit operating system.
[12] `http://vrp.atd-lab.inf.puc-rio.br`

7 Conclusion

The ACO seems like a good heuristic approach to deal with optimization problems. It should be remarked that although it performed quite well in our benchmark, the algorithm is really sensitive to its parameters. We had to tweak them a while until we got the results - I believe we would have got even better results if we had have spent more time with it. The second thing I would like to mention is the randomness that the ACO uses. We have used a static seed for the random number generator for every run. Once you use a different seed for each run, you would get results that occasionally vary in dozens of percent. So, it does not necessarily mean that if you have a proper configuration, you would get a good result on a first try.

References

[1] Norman Biggs. *Graph theory 1736-1936*. Oxford u.a: Clarendon Press, 1998. ISBN: 9780198539162.

[2] G. B. Dantzig and J. H. Ramser. „The Truck Dispatching Problem“. In: *Management Science* 6.1 (Oct. 1959), pp. 80–91. DOI: 10.1287/mnsc.6.1.80. URL: https://doi.org/10.1287%2Fmnsc.6.1.80.

[3] Gilbert Laporte. „The Traveling Salesman Problem, the Vehicle Routing Problem, and Their Impact on Combinatorial Optimization“. In: *International Journal of Strategic Decision Sciences* 1.2 (2010), pp. 82–92. DOI: 10.4018/jsds.2010040104. URL: https://doi.org/10.4018%2Fjsds.2010040104.

[4] Gilbert Laporte. „The Traveling Salesman Problem, the Vehicle Routing Problem, and Their Impact on Combinatorial Optimization“. In: *Decision Making Theories and Practices from Analysis to Strategy*. IGI Global, pp. 342–352. DOI: 10.4018/978-1-4666-1589-2.ch018. URL: https://doi.org/10.4018%2F978-1-4666-1589-2.ch018.

[5] J. K. Lenstra and A. H. G. Rinnooy Kan. „Complexity of vehicle routing and scheduling problems“. In: *Networks* 11.2 (1981), pp. 221–227. DOI: 10.1002/net.3230110211. URL: https://doi.org/10.1002%2Fnet.3230110211.

[6] Richa Bajaj and Vikas Malik. „A Review on Optimization with Ant Colony Algorithm“. In: *Journal of Network Communications and Emerging Technologies (JNCET) www. jncet. org* 6.7 (2016).

[7] B. Bullnheimer, R.F. Hartl, and C. Strauss. „An improved Ant System algorithm for theVehicle Routing Problem“. In: *Annals of Operations Research* 89.0 (1999), pp. 319–328. ISSN: 1572-9338. DOI: 10.1023/A:1018940026670. URL: http://dx.doi.org/10.1023/A:1018940026670.

[8] Gilbert Laporte and Frédéric Semet. „Classical heuristics for the capacitated VRP“. In: *The vehicle routing problem*. Society for Industrial and Applied Mathematics. 2001, pp. 109–128.

[9] David Pisinger and Stefan Ropke. „A general heuristic for vehicle routing problems“. In: *Computers & Operations Research* 34.8 (2007), pp. 2403–2435. ISSN: 0305-0548. DOI: http://doi.org/10.1016/j.cor.2005.09.012. URL: http://www.sciencedirect.com/science/article/pii/S0305054805003023.

[10] Jean-Francois Cordeau and Québec) Groupe d'études et de recherche en analyse des décisions (Montréal. *The VRP with time windows*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

[11] William Ho et al. „A hybrid genetic algorithm for the multi-depot vehicle routing problem“. In: *Engineering Applications of Artificial Intelligence* 21.4 (2008), pp. 548–557. ISSN: 0952-1976. DOI: http://doi.org/10.1016/j.engappai.2007.06.001. URL: http://www.sciencedirect.com/science/article/pii/S0952197607000887.

[12] Esmat Zare-Reisabadi and S. Hamid Mirmohammadi. „Site dependent vehicle routing problem with soft time window: Modeling and solution approach“. In: *Computers & Industrial Engineering* 90 (2015), pp. 177–185. ISSN: 0360-8352. DOI: http://doi.org/10.1016/j.cie.2015.09.002. URL: http://www.sciencedirect.com/science/article/pii/S0360835215003575.

[13] Alfredo Olivera and Omar Viera. „Adaptive memory programming for the vehicle routing problem with multiple trips“. In: *Computers & Operations Research* 34.1 (2007), pp. 28–47. ISSN: 0305-0548. DOI: http://doi.org/10.1016/j.cor.2005.02.044. URL: http://www.sciencedirect.com/science/article/pii/S0305054805001346.

[14] Guy Desaulniers and Québec) Groupe d'études et de recherche en analyse des décisions (Montréal. *The VRP with pickup and delivery*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

[15] Alvina G.H. Kek, Ruey Long Cheu, and Qiang Meng. „Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots“. In: *Mathematical and Computer Modelling* 47.1–2 (2008), pp. 140–152. ISSN: 0895-7177. DOI: http://doi.org/10.1016/j.mcm.2007.02.007. URL: http://www.sciencedirect.com/science/article/pii/S0895717707001215.

[16] Sandra Ulrich Ngueveu, Christian Prins, and Roberto Wolfler Calvo. „An Effective Memetic Algorithm for the Cumulative Capacitated Vehicle Routing Problem“. In: *Comput. Oper. Res.* 37.11 (Nov. 2010), pp. 1877–1885. ISSN: 0305-0548. DOI: 10.1016/j.cor.2009.06.014. URL: http://dx.doi.org/10.1016/j.cor.2009.06.014.

[17] P. Toth and D. Vigo. *Vehicle Routing*. Ed. by Daniele Vigo and Paolo Toth. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014. DOI: 10.1137/1.9781611973594. eprint: http://epubs.siam.org/doi/pdf/10.1137/1.9781611973594. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611973594.

[18] M. Dorigo and T. Stützle. *Ant Colony Optimization*. A Bradford book. BRADFORD BOOK, 2004. ISBN: 9780262042192. URL: https://www.amazon.com/Ant-Colony-Optimization-MIT-Press-ebook/dp/B004HHORGU.

[19] Xin-She Yang and Mehmet Karamanoglu. „1 - Swarm Intelligence and Bio-Inspired Computation: An Overview“. In: *Swarm Intelligence and Bio-Inspired Computation*. Ed. by Xin-She Yang et al. Oxford: Elsevier, 2013, pp. 3–23. ISBN: 978-0-12-405163-8. DOI: http://dx.doi.org/10.1016/B978-0-12-405163-8.00001-6. URL: http://www.sciencedirect.com/science/article/pii/B9780124051638000016.

[20] Renbin Xiao. „5 - Modeling and Simulation of Ant Colony's Labor Division: A Problem-Oriented Approach“. In: *Swarm Intelligence and Bio-Inspired Computation*. Ed. by Xin-She Yang et al. Oxford: Elsevier, 2013, pp. 103–135. ISBN: 978-0-12-405163-8. DOI: http://dx.doi.org/10.1016/B978-0-12-405163-8.00005-3. URL: http://www.sciencedirect.com/science/article/pii/B9780124051638000053.

[21] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. „The ant system: An autocatalytic optimizing process“. In: (1991).

[22] Alberto Colorni Marco Dorigo Vittorio Maniezzo. „Distributed optimization by ant colonies“. In: *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. Mit Press. 1992, p. 134.

[23] Christian Blum. „Ant colony optimization: Introduction and recent trends“. In: *Physics of Life Reviews* 2.4 (Dec. 2005), pp. 353–373. DOI: 10.1016/j.plrev.2005.10.001. URL: https://doi.org/10.1016%2Fj.plrev.2005.10.001.

[24] Silvia Mazzeo and Irene Loiseau. „An Ant Colony Algorithm for the Capacitated Vehicle Routing“. In: *Electronic Notes in Discrete Mathematics* 18 (2004), pp. 181–186. ISSN: 1571-0653. DOI: http://dx.doi.org/10.1016/j.endm.2004.06.029. URL: http://www.sciencedirect.com/science/article/pii/S1571065304010868.

[25] Thomas Stützle and Holger H. Hoos. „MAX-MIN Ant System“. In: *Future Gener. Comput. Syst.* 16.9 (June 2000), pp. 889–914. ISSN: 0167-739X. URL: http://dl.acm.org/citation.cfm?id=348599.348603.

[26] S. D. Shtovba. „Ant Algorithms: Theory and Applications“. In: *Programming and Computer Software* 31.4 (2005), pp. 167–178. ISSN: 1608-3261. DOI: 10.1007/s11086-005-0029-1. URL: http://dx.doi.org/10.1007/s11086-005-0029-1.

[27] John E. Bell and Patrick R. McMullen. „Ant colony optimization techniques for the vehicle routing problem“. In: *Advanced Engineering Informatics* 18.1 (2004), pp. 41–48. ISSN: 1474-0346. DOI: http://dx.doi.org/10.1016/j.aei.2004.07.001. URL: http://www.sciencedirect.com/science/article/pii/S1474034604000060.

[28] Bin Yu, Zhong-Zhen Yang, and Baozhen Yao. „An improved ant colony optimization for vehicle routing problem“. In: *European Journal of Operational Research* 196.1 (2009), pp. 171–176. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/j.ejor.2008.02.028. URL: http://www.sciencedirect.com/science/article/pii/S0377221708002373.

[29] Christian Nilsson. *Heuristics for the traveling salesman problem*. Tech. rep. Tech. Report, Linköping University, Sweden, 2003.

[30] Narihiro Park, Hiroyuki Okano, and Hiroshi Imai. „A path-exchange-type local search algorithm for vehicle routing and its efficient search strategy“. In: *in Journal of the Operations Research Society of Japan* (2000), pp. 197–208.

[31] James McCaffrey. *Ant Colony Optimization*. Feb. 2012. URL: https://msdn.microsoft.com/en-us/magazine/hh781027.aspx.

[32] Marcus Randall and Andrew Lewis. „A Parallel Implementation of Ant Colony Optimization“. In: *Journal of Parallel and Distributed Computing* 62.9 (2002), pp. 1421–1432. ISSN: 0743-7315. DOI: http://dx.doi.org/10.1006/jpdc.2002.1854. URL: http://www.sciencedirect.com/science/article/pii/S074373150291854X.

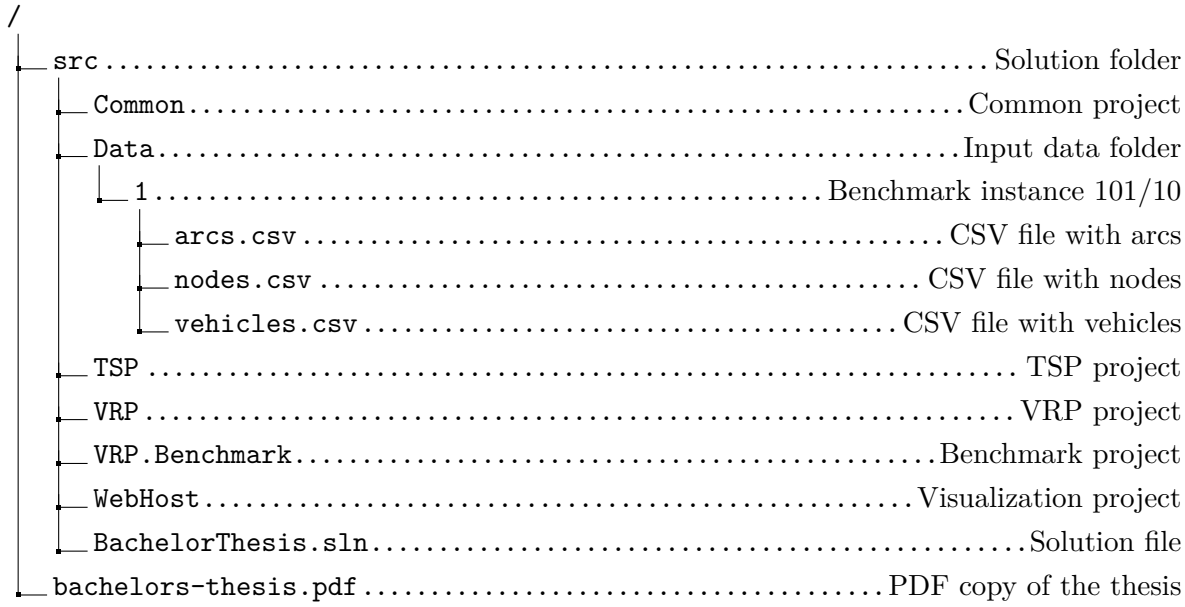
[33] Stefan Ropke and David Pisinger. „An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows“. In: *Transportation science* 40.4 (2006), pp. 455–472.

[34] Gerhard Schrimpf et al. „Record Breaking Optimization Results Using the Ruin and Recreate Principle“. In: *Journal of Computational Physics* 159.2 (2000), pp. 139–171. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1006/jcph.1999.6413. URL: http://www.sciencedirect.com/science/article/pii/S0021999199964136.

[35] Tomáš Režnar et al. „Probabilistic time-dependent vehicle routing problem“. In: *Central European Journal of Operations Research* (Oct. 2016). DOI: 10.1007/s10100-016-0459-2. URL: https://doi.org/10.1007%2Fs10100-016-0459-2.

A Contents of the enclosed CD

Directories and files of the enclosed CD are in the following structure:



B Compilation and run

The prerequisite is to have installed .NET Core SDK versions 1.0.0 and 1.1.0.

The first way involves the Visual Studio 2017. Open the solution (BachelorThesis.sln) and build/compile the desired project.

The second way is via Command Line Interface (CLI). The root folder of the solution consists of several folders (projects) - TSP, VRP, Webhost and VRP.Benchmark. Navigate to one of them and execute the following commands:

```
> dotnet restore
> dotnet build
> dotnet run
```

Alternatively, you can restore and build all the projects at once, just execute dotnet restore and dotnet build in the root folder. To run a project from the solution directory, you have to specify which project you would like to run:

```
> dotnet restore
> dotnet build
> dotnet run -p ProjectName/ProjectName.csproj
(change ProjectName to one of following TSP | VRP | WebHost | VRP.Benchmark)
```